ZMW AP

OIPE IAP85
AUG 1 7 2006
PATENT & TRADEMARK OFFICE

Docket No. 1363-004 Patent

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of		:
		:
	NEIL W. TAYLOR	:
		:
Serial No.:	09/862,828	: Group Art Unit: 2135
		:
Filed:	May 22, 2001	: Examiner: Son, Linh L D

For: METHODS FOR DETECTING EXECUTABLE CODE WHICH HAS BEEN ALTERED

APPEAL BRIEF

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Responsive to the Office Action of April 6, 2006 and the Notice of Panel Decision from Pre-Appeal Brief Review of July 20, 2006, the Applicant hereby appeals the final rejection of claims 1-4 and 7-22. Also, a fee transmittal indicating payment of the Appeal Brief Fee in the amount of $500.00 accompanies this paper according to 37 C.F.R. §41.20(b)(2). A Notice of Appeal and attendant Notice of Appeal fee, in the amount of $500.00 according to 37 C.F.R. §§41.20(b)(1), was earlier filed on May 8, 2006 along with a Pre-Appeal Brief Request for Review. It is believed no additional fees are due.

08/17/2006 HGUTEMA1 00000057 110978 09862828
01 FC:1402 500.00 DA

I. Real Party in Interest

The real party in interest is Novell, Inc., a corporation of the State of Delaware, having a principal place of business at 1800 South Novell Place, Provo, Utah 84606.

II. Related Appeals and Interferences

The Appellant knows of no other prior or pending appeals, interferences, or judicial proceedings, which may be related to, directly affect, or be directly affected by, or have a bearing on, the Board's decision in this Appeal.

III. Status of Claims

All pending claims (1-4 and 7-22) stand finally rejected under 35 U.S.C. §103(a) as obvious over Slivka et al. U.S. Patent No. 5,493,649 in view of Angelo U.S. Patent No. 5,944,821. Claims 5 and 6, on the other hand, have long ago been canceled. On appeal, the Appellant traverses the rejections of all pending claims. Claims 1, 8, 16, 21 and 22 are independent.

IV. Status of Amendments

No amendment has been filed subsequent to the Final Office Action dated April 6, 2006 and all previous amendments have been entered. The form of the claims for purposes of appeal are those presented in the Amendment and Remarks filed by the Appellant on January 16, 2006 (received by the Patent Office on January 19, 2006, as indicated by the date

stamp on the return postcard). As required, a copy of the claims is included herewith in Appendix form with double-spacing format.

V. Summary of Claimed Subject Matter

Claims 1-4 and 7-22 are pending. Claims 1, 8, 16, 21 and 22 are independent. The independent claims and claims 11, 12 and 19 are argued particularly relative to the cited art.

The present invention relates broadly to computers, especially executable code able to be validated as unmodified or unaltered. More particularly, the present invention contemplates loading executable code in resident memory of an operating system and calculating an initial score indicative of an unaltered size or format of the code. The score is also calculated upon an initial loading of the code into memory, or shortly thereafter. One of randomly or substantially each time the executable code is thereafter launched for use, a plurality of subsequent scores indicative of the size or format of the code are calculated. To determine modification or alteration of the code, the plural subsequent scores are compared exclusively to the initial score and to no others. If the scores match or do not vary, the code is deemed original. If the scores do not match or vary, the code is deemed modified in either size or format. In this manner, "software distributed and [] supported by the software vendor may be continually validated for authenticity." *Appellant's Specification,* p. 4, l. 22 - p. 5, l. 2. It is also the case that "alterations are detectable and reportable to the software vendor." *Id.* at p. 5, l. 2.

As indicated throughout the Appellant's Background of the Invention section, it is intensely appreciated that software hackers have become adept over the years and now possess exceptional talent for inserting malicious code and/or viruses into executable code. To this end, the instant invention contemplates requiring calculating plural subsequent scores

nearly "each time"[1] the "executable code is launched for use" after the initial loading of the executable code. Alternatively, the instant invention contemplates calculating plural subsequent scores "randomly."[2] In this manner, hacker attempts on the executable code are thwarted by either being tedious or unpredictable. In some instances, the instant invention goes so far as to contemplate doubly performing calculations of subsequent scores both "randomly and substantially each time the executable code is launched for use." *Appellant's Specification*, claims 11 and 12. As another labyrinth of hacker protection, the doubly secure methodology is supplemented by requiring calculation of subsequent scores with "predetermined time intervals." *Appellant's Specification*, claim 12.

In general, "executable code" of the invention is that which is typically "translate[d from] a source code" by a "compiler." *Appellant's Specification*, p. 7, ll. 11-12. It is also code "not usually readable to humans, although adept programmers have developed methods of reading and understanding at least some portions." *Appellant's Specification*, p. 7, ll. 17-19. Distribution of the code occurs in a variety of ways including, for example, "direct download, on a separate computer readable medium for installation, and others." *Appellant's Specification*, p. 8, ll. 19-20. It may also be found in an "encrypted format, or in a source code format, and other formats." *Appellant's Specification*, p. 8, ll. 21-22. In some instances, "interpretive languages, such as PERL, and others," software embodies "a format which is both its source format and its executable format." *Appellant's Specification*, p. 8, l.. 24 - p. 9, l. 2. Altogether, executable code of the invention is "any format of the software (e.g., program instructions, data, scripts, control files, parameters, and the like) which is used for execution on a computing device." *Appellant's Specification*, p. 9, ll. 3-4. Upon installation, it may be representatively found in "Novell's NetWare operating system . . . [or]

---

[1]Support for these limitations are found throughout the specification. Especially, *p. 15, ll. 9-18*.

[2] *Id.*

other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. Alternatively stated, executable code "(e.g., program instructions and data used by the instructions) resid[es] in the resident memory of operation system." *Appellant's Specification*, p. 4, ll. 17-18.

Alterations or modifications of the executable code, on the other hand, occur when a party is "permitted to bypass important licensing aspect's of the vendor's products." *Appellant's Specification*, p. 7, ll. 23-24. It may also take the form of "insert[ion] of computer viruses . . . which may substantially damage the reputation of the vendor and potentially damage non vendor software on a consumer's computing device." *Appellant's Specification*, p. 7, l. 24 - p. 8, l. 3. In still other instances, "[a]lterations may include, by way of example only, the bypassing of automatic licensing checks, the insertion of malicious computer viruses, and others." *Appellant's Specification*, p. 1, ll. 8-10. Modified code also enables "hackers [to] create a permanent fix by patching the operating system or the executable code itself." *Appellant's Specification*, p. 2, ll. 8-9. It also may be modified according to "the image of the executable which is resident in the memory of the operating system." *Appellant's Specification*, p. 2, ll. 22-23. By inference then, unmodified or unaltered executable code is of the type previously described lacking the attributes of modification or alteration.

In the Figures, the flow diagrams relate to: "validating executable code" (Figure 1); "disabling executable code which has been altered" (Figure 2); and "authenticating executable code" (Figure 3). *Appellant's Specification*, p. 6, ll. 13-16.

In independent claim 1, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are <u>representatively</u> found in the specification at the parenthetical cite as follows:

1. **A method for validating executable code resident in an operating system having executable instructions** ("Fig. 1 depicts . . . a method of validating executable code," *Appellant's Specification*, p. 6, l. 13; and executable code "resid[es] in the resident

memory of the operating system," for example. *Appellant's Specification*, p. 4, ll. 17-18.), comprising the steps of:

identifying an executable code having an unaltered size ("Fig. 1 identifies an executable code in step 10." *Appellant's Specification*, p. 11, l. 10);

calculating an initial score associated with the executable code when the executable code is initially or shortly thereafter loaded into an operating system ("Either during the installation of the executable code or shortly after the initial installation of the executable code, a set of executable instructions is run against the distributed code to generate a score associated with the executable code." *Appellant's Specification*, p. 9, ll. 5-8; *see also*, Figure 1, step 20, "Score on load" with the score occurring on the identified code of step 10, Figure 1, "which is to be installed, or as very recently been installed." *Appellant's Specification*, p. 11, ll. 10-11.);

saving the initial score ("[t]he score may then be stored in step 30 [Figure 1]." *Appellant's Specification*, p. 11, l. 14.);

one of randomly and substantially each time the executable code is launched for use, calculating a plurality of subsequent scores on the executable code ("a subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18.);

exclusively comparing each of the subsequent scores to the saved initial score and to no other scores ("Once a subsequent score is obtained, the initially generated score associated with the executable code is compared to the newly acquired executable code in step 60 [Figure 1]." *Appellant's Specification*, p. 12, ll. 17-19. In other words, the

subsequent scores are not compared to other subsequent scores, but to the initial score. Also, "[t]he initial score may be compared to subsequent computed scores randomly, periodically, or by manual selection of a user." *Appellant's Specification*, p. 4, ll. 10-11.);

if the each of the subsequent scores do not vary from the saved initial score, concluding the executable code maintains the unaltered size ("If the scores are not identical, then the executable code has been altered in some way since the initial score was taken." *Appellant's Specification*, p. 12, ll. 19-20. Inferentially, if the scores are the same, then the executable code has not been altered. It "is extremely unlikely that the data associated with the executable code may have been altered and yet still generate the same score as the initial score." *Appellant's Specification*, p. 9, l. 24 - p. 10, l. 2.); and

if any of the subsequent scores vary from the saved score, concluding the executable code has an altered size ("If the scores are not identical, then the executable code has been altered in some way since the initial score was taken." *Appellant's Specification*, p. 12, ll. 19-20.).

In independent claim 8, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are representatively found in the specification at the parenthetical cite as follows:

8. **A method for disabling executable code which has been modified without authorization having executable instructions** ("Fig. 2 depicts a flow diagram of a method for disabling executable code which has been altered." *Appellant's Specification*, p. 6, ll. 14-15.), comprising the steps of:

identifying an executable code in an operating system having an unaltered format ("Initially a set of executable instructions depicted by Fig. 2 receive a score in step 80 associated with a specific executable code." *Appellant's Specification*, p. 13, ll. 16-18. Upon installation, executable code may be representatively found in "Novell's NetWare

operating system . . . [or] other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. Alternatively stated, executable code "(e.g., program instructions and data used by the instructions) resid[es] in the resident memory of operation system." *Appellant's Specification*, p. 4, ll. 17-18.);

calculating a score associated with the executable code exclusively within an operating system of a computing device independent of a system management mode of operation (*Id.* Also, "a set of executable instructions is run against the distributed executable code to generate a score associated with the executable score." *Appellant's Specification*, p. 9, ll. 6-7. Upon installation, executable code may be representatively found in "Novell's NetWare operating system . . . [or] other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. "In an exemplary embodiment a set of executable code responsible for performing one or more calculations against the executable code resides as one or more low level routines within the operating system." *Appellant's Specification*, p. 10, ll. 12-14. Stated negatively, that which is located as low level routines within the operating system for making calculations need not reside in a peculiar location or a peculiar mode of operation such as a system management mode of operation.);

one of randomly and substantially each time the executable code is launched for use, calculating subsequent scores associated with the executable code ("a subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18.);

determining the executable code has an altered format if the score is not equal to any of the subsequent scores, the determining exclusively including comparing each

of the subsequent scores to the score with no other score comparisons occurring (Including Steps 100, 110, Figure 2. "If the scores are not identical, then the executable code has been altered in some way since the initial score was taken." *Appellant's Specification*, p. 12, ll. 19-20. Inferentially, if the scores are the same, then the executable code has not been altered. It "is extremely unlikely that the data associated with the executable code may have been altered and yet still generate the same score as the initial score." *Appellant's Specification*, p. 9, l. 24 - p. 10, l. 2. "Once a subsequent score is obtained, the initially generated score associated with the executable code is compared to the newly acquired executable code in step 60 [Figure 1]." *Appellant's Specification*, p. 12, ll. 17-19. In other words, the subsequent scores are not compared to other subsequent scores, but to the initial score. Also, "[t]he initial score may be compared to subsequent computed scores randomly, periodically, or by manual selection of a user." *Appellant's Specification*, p. 4, ll. 10-11.); and

disabling the executable code if the score is not equal to any of the subsequent scores ("If the initial score and the subsequent score vary, a determination is made as to whether the variation is a valid condition (e.g. vendor authorized), in which case a new initial score is saved. If the variation in the scores is not valid, the executable code is disabled in step 140. *Appellant's Specification*, p. 14, ll. 11-13.).

In dependent claims 11, 12 and 19, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are representatively found in the specification at the parenthetical cite as follows:

11. The method of claim 8, **wherein the subsequent scores are calculated randomly and substantially each time the executable code is launched for use.**

12. **The method of claim 11, wherein the subsequent scores are further calculated at one or more predetermined time intervals.**

19. The method of claim 16, **wherein the subsequent scores are received each time the executable code is initiated in the memory for an execution.**

("[A] subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18. Also, "[t]he initial score may be compared to subsequent computed scores randomly, periodically, or by manual selection of a user." *Appellant's Specification*, p. 4, ll. 10-11. To this end, the base claims upon which the above dependent claims depend contemplate requiring calculating plural subsequent scores nearly each time the executable code is launched for use after the initial loading of the executable code. Alternatively, they require calculating plural subsequent scores randomly. In this manner, hacker attempts on the executable code are thwarted by either being tedious or unpredictable. In some instances, claims 12 and 13 the instant invention goes so far as to contemplate doubly performing calculations of subsequent scores both "randomly <u>and</u> substantially each time the executable code is launched for use." As another labyrinth of hacker protection inn claim 13, the doubly secure methodology is supplemented by requiring calculation of subsequent scores with "predetermined time intervals." Claim 19, on the other hand, requires subsequent scores to be received "each time" or every time the executable code is initiated in the memory for an execution. "Moreover, a combination of comparison and score generations may occur." *Appellant's Specification*, p. 15, ll. 11-12.)

In independent claim 16, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are <u>representatively</u> found in the specification at the parenthetical cite as follows:

16. **A method of authenticating executable code resident in a memory having executable instructions** ("Fig. 3 depicts one embodiment for a flow diagram of a method of authenticating executable code." *Appellant's Specification*, p. 14, ll. 19-20. Upon installation, executable code may be representatively found in "Novell's NetWare operating system . . . [or] other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. Alternatively stated, executable code "(e.g., program instructions and data used by the instructions) resid[es] in the resident memory of operation system." *Appellant's Specification*, p. 4, ll. 17-18.), comprising the steps of:

identifying an executable code having an unaltered format (step 10, identifies executable code. By inference from earlier statements, unmodified or unaltered executable code is of the type previously described lacking the attributes of modification or alteration.);

acquiring a score associated with an executable code which was established when the executable code was first loaded into a memory of an operating system ("An initial score is acquired in step 170, the score being associated with a specific executable code and acquired by performing a calculation against the executable code when the executable code was initially loaded or shortly thereafter. . ." *Appellant's Specification*, p. 14, ll. 20-24.);

one of randomly and substantially each time the executable code is launched for use after the executable code is said first loaded, receiving subsequent scores on the executable code while the executable code is in the memory ("a subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18.);

exclusively comparing the subsequent scores to the score with neither the subsequent scores nor the score being compared to any other values ("Sometime after an initial score is obtained, one or more subsequent scores are acquired in step 190. In step 180, the initial score and the subsequent scores are compared." *Appellant's Specification*, p. 15, ll. 3-5.);

if the subsequent scores do not vary from the score, executing the executable code in the unaltered format (If the compared scores are "different (step 210) the executable code is suspended from operation or disabled in step 220." *Appellant's Specification*, p. 15, ll. 5-6. By inference, if compared scores are the same or do not vary, they need not be disabled or suspended from operation. Rather, they will be used for their intended purpose); and

if any of the subsequent scores vary from the saved score, indicating the executable code has an altered format (If the compared scores are different "(step 210)"... the failed comparison is reported." *Appellant's Specification*, p. 15, ll. 5-6).

In independent claim 21, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are representatively found in the specification at the parenthetical cite as follows:

21. **Functional data used to validate executable code embodied in a computer readable medium, the data comprising:**

an unaltered original format ("Fig. 1 identifies an executable code in step 10" *Appellant's Specification*, p. 11, l. 10. Executable code lacking the indicia of alteration or modification, from above, is then that which is unaltered or unmodified.);

a first score associated with an executable code when the executable code is initially loaded into an operating system ("Either during the installation of the executable code or shortly after the initial installation of the executable code, a set of executable

instructions is run against the distributed code to generate a score associated with the executable code." *Appellant's Specification*, p. 9, ll. 5-8; *see also*, Figure 1, step 20, "Score on load" with the score occurring on the identified code of step 10, Figure 1, "which is to be installed, or as very recently been installed." *Appellant's Specification*, p. 11, ll. 10-11. Upon installation, executable code may be representatively found in "Novell's NetWare operating system . . . [or] other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. Alternatively stated, executable code "(e.g., program instructions and data used by the instructions) resid[es] in the resident memory of operation system." *Appellant's Specification*, p. 4, ll. 17-18.);

a plurality of second scores associated with the executable code at one of 1) a random period of time subsequent to when the executable code was initially loaded and 2) substantially each time the executable code is launched for use subsequent to when the executable code was initially loaded and each second score being operable to be exclusively compared with the first score to determine if the executable code has been altered since the initial load ("a subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18. "Once a subsequent score is obtained, the initially generated score associated with the executable code is compared to the newly acquired executable code in step 60 [Figure 1]." *Appellant's Specification*, p. 12, ll. 17-19. In other words, the subsequent scores are not compared to other subsequent scores, but to the initial score. Also, "[t]he initial score may be compared to subsequent computed scores randomly, periodically, or by manual selection of a user."

Appellant's Specification, p. 4, ll. 10-11. Also, "a combination of comparisons and score generations may occur." *Appellant's Specification*, p. 15, ll. 11-12.); and

an altered unoriginal format if the first and second scores do not equal ("If the scores are not identical, then the executable code has been altered in some way since the initial score was taken." *Appellant's Specification*, p. 12, ll. 19-20.).

In independent claim 22, and consistent with Figures 1-3, for example, the below-limitations of the claim, in **bold**, are representatively found in the specification at the parenthetical cite as follows:

22. **A system for validating executable code, comprising:**

an executable code having an unaltered size ("Fig. 1 identifies an executable code in step 10" *Appellant's Specification*, p. 11, l. 10. Executable code lacking the indicia of alteration or modification, from above, is then that which is unaltered or unmodified.);

a scoring set of executable instructions operable to receive and record a score associated with the executable code when the code is initially loaded into a computer readable medium ("Either during the installation of the executable code or shortly after the initial installation of the executable code, a set of executable instructions is run against the distributed code to generate a score associated with the executable code." *Appellant's Specification*, p. 9, ll. 5-8; *see also*, Figure 1, step 20, "Score on load" with the score occurring on the identified code of step 10, Figure 1, "which is to be installed, or as very recently been installed." *Appellant's Specification*, p. 11, ll. 10-11. Upon installation, executable code may be representatively found in "Novell's NetWare operating system . . . [or] other operating systems." *Appellant's Specification*, p. 6, ll. 20-21. Alternatively stated, executable code "(e.g., program instructions and data used by the instructions) resid[es] in the resident memory of operation system." *Appellant's Specification*, p. 4, ll. 17-18. Also, "[s]oftware is developed, distributed, and executed in a variety of different formats."

Appellant's Specification, p. 7, ll. 1-2. Software is representatively distributed "on a separate computer readable medium." *Appellant's Specification*, p. 8, l. 19.);

a comparing set of executable instructions in an operating system of a computing device independent of a system management mode of operation operable to receive subsequent scores associated with the code and to one of randomly and substantially each time the executable code is launched for use exclusively compare the score and the subsequent scores to determine if the code has been altered ("a subsequent score may be acquired each time the executable code is launched for use and then may also be acquired randomly or periodically while the executable code is in use. In this way, any alteration to the executable code, after its initial launch may be detected while the executable code is in use. As one skilled in the art will readily appreciate, this will permit changes to the image of the executable code to be detected while the executable code is executing." *Appellant's Specification*, p. 15, ll. 12-18. "Once a subsequent score is obtained, the initially generated score associated with the executable code is compared to the newly acquired executable code in step 60 [Figure 1]." *Appellant's Specification*, p. 12, ll. 17-19. In other words, the subsequent scores are not compared to other subsequent scores, but to the initial score. Also, "[t]he initial score may be compared to subsequent computed scores randomly, periodically, or by manual selection of a user." *Appellant's Specification*, p. 4, ll. 10-11. Also, "a combination of comparisons and score generations may occur." *Appellant's Specification*, p. 15, ll. 11-12. Even further, "a set of executable instructions is run against the distributed executable code to generate a score associated with the executable score." *Appellant's Specification*, p. 9, ll. 6-7. "In an exemplary embodiment a set of executable code responsible for performing one or more calculations against the executable code resides as one or more low level routines within the operating system." *Appellant's Specification*, p. 10, ll. 12-14. Stated negatively, that which is located as low level routines within the

operating system for making calculations need not reside in a peculiar location or a peculiar mode of operation such as a system management mode of operation.); and

an executable code having an altered size if the score and the subsequent scores do not equal ("If the scores are not identical, then the executable code has been altered in some way since the initial score was taken." *Appellant's Specification*, p. 12, ll. 19-20.).

VI. Grounds of Rejection to be Reviewed on Appeal

The Board must determine whether claims 1-4 and 7-22 are rendered obvious under 35 U.S.C. §103(a) over Slivka in view of Angelo. In this regard, the Board must essentially determine whether the references teach what the Examiner suggests they do. Namely, it should be determined: a) whether Angelo supplies the missing teaching of Slivka; b) whether Slivka's excessive delay in checksum comparisons is remedied by Angelo's program hash routines during safestart modes of operation prior to loading of programs into memory; c) whether Angelo discloses scores or calculations of scores associated with executable code at a time when the code is "initially or shortly thereafter loaded into an operating system."[3] or whether Angelo concerns itself with calculating hash values of programs prior to or before the program being loaded into memory and executed; d) whether Slivka and Angelo are properly combined; and e) whether the Examiner has met his *prima facie* burden.

To the extent the Board's determination finds any of the above in favor of the Appellant, the entirety of the claims should be adjudicated patentable in view of the pending rejections.

[3] This claim language is expressly found in claim 1. Of course, the other claims have related language but their slight variations change the overall scope of the claims. In prior papers, the Applicant also touts aspects of the instant invention as including this timing of when the code is loaded.

VII. Argument

A. Slivka teaches either "code" or underlying "data" checksums. The underlying data checksums do not apply to the instant invention because they are not "executable code." The "code" checksums, on the other hand, are greatly delayed from occurring multiple times according to "expiration of a periodic interval." They are not "randomly and substantially each time" executed, as admitted by the Examiner, and Angelo does not supply the missing teaching. Angelo concerns itself with program hash routines "prior to or before" memory loading, not executable code "initially or shortly thereafter" loaded into an operating system.

SLIVKA:

Slivka teaches a comparison of one or more checksums. However, Slivka's checksums relate either to code or data. In turn, code and data checksums are calculated/performed relative to code sections and data sections, respectively, of computer programs 310, 312, 314, an MS-DOS operating system 304 and/or a file management component 308. As defined, "[t]he code section of a computer program contains the computer instructions that operate upon the data of the computer program." *Col. 3, ll. 1-3.* On the other hand, "[t]he data section of the computer program contains the data that the instructions of the computer program use for operation." *Col. 3, ll. 3-5.* Further, code and data are precisely known as separate functional sections. For instance, Slivka teaches, in the context of the file management component, that "when performing an operation, [the file management component] always executes code as well as either reading from or writing to the data section." *Col. 3, ll. 37-39.* Accordingly, only the checksums of the code section of Slivka are relevant to the instant invention. That is, the instant invention precisely recites the validation of "executable code" and relates not to underlying data.

With this in mind, Slivka calculates a checksum for the "code section" at step 402. Of Figure 4A. Later, at step 410, a "new code checksum" is calculated. In the event the first and new "code" checksums are not equivalent, corruption is indicated at step 428 of Figure 4B, indicated by off-page connector A. Then, "processing ends, before the code is executed." *Col. 3, ll. 57-58.* As is clear, this teaches a checksum comparison that is wholly a one-to-one comparison of a first checksum to a second or new checksum. It is unequivocally never the calculation of "an initial score" and a calculation of subsequent "plural scores," much less subsequent "exclusive" comparison of "each" of the subsequent "plural" scores to the initial score "and to no other scores," as representatively claimed in the Appellant's independent claim 1. In other words, Slivka's one-to-one comparison of a first checksum to a second checksum does not have multiple instances of subsequent checksums, nor does it have exclusive comparisons between the multiple instances to the first checksum and to no others.

To the extent the code checksums are equivalent at step 412, "processing continues." *Col. 3, l. 59.* In this regard, the checksums of the underlying "data" checksums, not "code" checksums, are invoked at step 416 and comparisons made at step 418. If the two are not equivalent, "the file management component indicates to the computer that the <u>data section</u> of the file management component has been corrupted and processing ends." *Underlining added, Col. 3, l. 67 - col. 4, l. 3.* Upon close inspection, the latest comparison of checksums is that of a first and second "data checksum" being executed. Relative to the claims, this is first and foremost not a checksum at all concerned with "executable code," but to underlying data. For at least this reason, it is irrelevant. Second, the first and second "data checksums" make another instance of a wholly one-to-one comparison of a first checksum to a second or new checksum. It is unequivocally never the calculation of "an initial score" and a calculation of subsequent "plural scores," much less subsequent "exclusive" comparison of "each" of the subsequent "plural" scores to the initial score "and to no other scores," as

representatively claimed in the Appellant's independent claim 1. In other words, one-to-one comparison of a first "data" checksum to a second "data" checksum does not have multiple instances of subsequent checksums, nor does it have exclusive comparisons between the multiple instances to the first checksum and to no others.

Continuing, if the two "data checksums" are equivalent at step 418, "processing continues." *Col. 4, ll. 3-4.* At step 420, access to the "data" is ascertained relative to read or write operations. If it is a "write" operation (alternatively stated as not a "read" operation), "an incremental checksum is calculated for the data section by invoking the incremental checksum routine." *Underlining added, Col. 4, ll. 7-8.* If, on the other hand, it is "read" operation, no invocation of an incremental checksum will be implemented because "the data section will not be modified." *Col. 4, ll. 9-10.* In either, the data is then accessed, step 422, and the processing "continues to step 406 to receive a requested operation," indicated by the off-page connector C. *Col. 4, ll. 14-15.*

As a result, a first iteration of Slivka's checksum processing is complete. In substance, it includes a single instance of comparing one "code" checksum to a second "code" checksum or a single instance of comparing one "data" checksum to a second "data" checksum. In either, both are wholly one-to-one comparisons of checksums and never the calculation of "an initial score" and a calculation of subsequent "plural scores," much less subsequent "exclusive" comparison of "each" of the subsequent "plural" scores to the initial score "and to no other scores," as representatively claimed in the Appellant's independent claim 1. In other words, one-to-one comparison of a first checksum to a second checksum does not have multiple instances of subsequent checksums, nor does it have exclusive comparisons between the multiple instances to the first checksum and to no others.

To the extent Slivka makes other, incremental "data" checksums at step 426, for instance, this relates to writing operations. Subsequent checksum comparisons are then made back to the immediately one prior checksum, not the original or first data section checksum.

Slivka particularly points this feature out as "the incremental checksum routine [that] adjusts the checksum **to account for a subsequent modification** by subtracting the data to be overwritten from the checksum and adding the overwriting ("new") data to the checksum." *Emphasis added, col. 4, ll. 55-59.* Stated more simply, the incremental checksum method "**recalculates** the first checksum so as to reflect the subsequent changes to the computer program after the computer program has been executed." *Emphasis added, col. 2, ll. 2-5.* In turn, this data section checksum feature of Slivka cannot implicate claims precisely requiring "exclusive" comparisons of plural subsequent scores back to an "initial" score and "to no other scores." *See, e.g., Applicant's independent claim 1.* ***Again, however, the Appellant mentions the data checksums are irrelevant to the instance invention because they are not executable code. Their description, however, is simply presented for completeness.***

To the extent Slivka calculates a third or more "code section" checksum (indicated in one instance by off-page connector "C" being introduced back into the flow diagram of Figure 4a before step 406), this third or more code section checksum, as well as the "new" or second code section checksum at step 410, is analyzed at step 408 to determine "whether a periodic interval has elapsed." *Col. 3, ll. 44-45.* According to Slivka, a periodic interval "can be based on timing, count of operations requested, or other suitable events." *Col. 3, ll. 45-46.* Purportedly, "modifications to the code section are **very rare**." *Emphasis added, col. 3, l. 40.* In turn, it is preferred that code section checksums not be compared or "checked upon receipt of every operation request." *Col. 3, ll. 42-43.* In other words, Slivka believes the rarity of changes to code sections of programs, for example, enables a delay to be introduced in the code section checksum comparison routine and it is a best practice to prevent code section checksum comparisons from occurring all the time. Otherwise, processing would bog down. Expressly, the "periodic interval" routine is the mechanism for

introducing delay and such is regular or periodic during all code section checksum comparisons.

All Claims:

In great contrast, all claims of the instant invention require calculating plural subsequent scores nearly "each time" the "executable code is launched for use" after the initial loading of the executable code. Alternatively, all claims of the instant invention require calculating plural subsequent scores "randomly." In this manner, the Applicant's invention contemplates thwarting hacker attempts on executable code by either being tedious or unpredictable. As indicated throughout the Applicant's Background of the Invention section, it is intensely appreciated that hackers have become adept over the years and now possess exceptional talent for inserting malicious code and/or viruses into executable code. Whereas, heretofore, they were essentially unconcerned or novices at it. To wit, Slivka's teaching characterizes alterations or modifications to code as "very rare" and thus only worthy of receiving "periodic interval" checksum comparisons. The Applicant's claims, however, are not so precluded. Rather, the Applicant's invention far exceeds Slivka's contemplated hacker prevention and all claims are submitted as allowable.

Claims 11, 12 and 19:

Additionally, the Applicant's claims 11 and 12 precisely recite that calculations of subsequent scores are doubly performed both "randomly and substantially each time the executable code is launched for use." Nowhere does Slivka contemplate or even appreciate this sophisticated doubly secure methodology. Claim 12 recites another labyrinth of protection by requiring calculation of plural subsequent scores with "predetermined time intervals" layered upon the doubly secure calculations of both "randomly and substantially each time the executable code is launched for use." Claim 19 even goes so far as to require subsequent scores "each time" the executable code is launched for use. In other words,

subsequent scores need be received "every time" the executable code is launched for use. Nowhere does Slivka's teaching approach this level of sophistication.

The Examiner also admits this by stating "Slivka does not specifically disclose the limitation of 'one of randomly and substantially each time the executable code is launched for use, calculating a plurality of subsequent scores on the executable code.'" *Underlining in original, 4-6-2006 Final Office Action*, p. 3, second paragraph. Relative to claims 11 and 12, however, the Examiner changes the language of the claims and recites "Slivka discloses ... 'receiving one or more additional scores periodically on the executable code and disabling the executable code if any of the subsequent score [sic] is not equal." *4-6-2006 Final Office Action*, p. 5, final three lines. As is clear, these claims do not recite what the Examiner says they do and are submitted as allowable according to the above. Relative to claim 19, the Examiner contends Slivka, indeed, discloses subsequent scores being received "each time" the code is used. *4-6-2006 Final Office Action*, p. 6, paragraph numbered 14. Clearly, this statement contradicts his early quoted statement at the beginning of this paragraph. It cannot be that Slivka avoids teaching "random and substantially each time" while at the same time teaching "each time." They are mutually exclusive and Slivka's periodic delay element makes his claim 19 argument flawed. For at least this reason, claim 19 is submitted as allowable.

ANGELO:

Angelo, on the other hand, does not supply the missing teaching rendering the claims obvious. Rather, Angelo concerns itself with calculating hash values of programs prior to or before the program being loaded into memory and executed. Angelo never discloses scores or calculations of scores associated with executable code at a time when the code is "initially or shortly thereafter loaded into an operating system."[4]

[4] This claim language is expressly found in claim 1. Of course, the other claims have related language but their slight variations change the overall scope of the claims. In prior papers, the Applicant also touts aspects of the instant

In all embodiments, Angelo's hash value comparisons occur at a time prior to the executable code being installed in an operating system. This is because Angelo concerns itself with a SAFESTART mode of operation and wants to be certain of assessing a file's integrity before it becomes loaded into an operating system to avoid operating system corruption. Representatively, Angelo's Figure 3 shows generating a SMI (System Management Interrupt, step 300); entering the SMM (System Management Mode of operation) via the CPU (step 302); generating a hash value (step 304); validating the stored hash table (step 306); checking entries in the hash table and verifying same (steps 308-316). Thereafter, it loads the program into memory and executes it (step 318). From the usage of antecedent language in the steps, it should be appreciated that step 304 requires the generation of a secure hash value for a "program to be executed." Then, at step 318, it is this same "program to be executed" that is subsequently "loaded into memory and executed." At *col. 9, ll. 6-8*, relative to Figure 2, Angelo reiterates this concept by stating the hash algorithm "securely register[s] and verif[ies] the integrity of software applications prior to execution." *Underlining added.* In other words, Angelo teaches hash value generation and comparison at a time prior to or before loading executable code (having its hash value generated) into an operating system. This, the Applicant respectfully submits, cannot then as a matter of law be used in rejecting claims requiring the contrary. *See also, Applicant's prior paper entitled AF-Request for Reconsideration, dated June 14, 2005.*

B. Angelo disparages both Slivka and the instant invention. The combination fails.

In the alternative, Angelo disparages solutions that attempt to solve problems of code modification detection via software products and/or in other than protected memory areas,

---

invention as including this timing of when the code is loaded.

such as SMM memory. For example,

> [i]n some earlier systems, a secure hash value is calculated and stored for newly installed software. Thereafter, when the computer is turned on again, the stored hash value is compared to a newly calculated value. If a discrepancy is found, the user is alerted. A main disadvantage with this method is that the integrity assessment codes must be stored on the hard disk, thus making the codes themselves susceptible to attack by malicious code. Reverse-engineering a modification detection code, while difficult, is not a mathematically intractable problem. Thus, software-only protective products can offer only limited insurance against the attack of malicious code, due mainly to architectural weakness present in most computer systems. A potential solution is to embed the modification detection code in a permanent read-only memory device, but this can make system reconfiguration quite difficult. *Col. 2, l. 60 - col. 3, l. 8.*

Thus, Angelo teaches away from the instant invention, especially in instances, such as **claims 21 and 22**, clearly requiring computer readable medium. Angelo teaches away from Slivka because Slivka clearly operates in memory 202 other than protected memory. The combination thusly fails. **Claims 8-16 and 22** also precisely state that the instant invention occurs "independent of a system management mode of operation." As is often the situation, software products desire operation as part and parcel of regular or routine modes of operation. For example, word processing, spread sheets, drawing, and other well known software products operate fully and independently of SAFESTART and other related modes of operation. To this end, at least claims 8-16 and 22 are allowable.

C. The Slivka and Angelo combination is based solely on hindsight reconstruction of the claims.

Along with the RCE filed in August 2005, the Appellant successfully argued past the

teachings of Angelo. Now, with the claims even more precisely defined and facing earlier anticipation rejections from Slivka, the Examiner returns to Angelo in his final rejection to piecemeal together an obviousness rejection. He also selectively culls bits and pieces of the references, without motivation to do so, to fit the claims as he seemingly, whimsically sees fit.

As the law has long held, the proper test of obviousness is whether the differences between the invention and the prior art are such that "the subject matter as a whole would have been obvious at the time the invention was made" to a person skilled in the art. *Stratoflex Inc. V. Aeroquip Corp.*, 713 F.2d 1530, 1538 (Fed. Cir. 1983)(Underlining added). Bear in mind, the Applicant originally filed this application on May 22, 2001. It is now over five full years since filing. The Applicant also reminds of the caution expressed by the Court of Appeals for the Federal Circuit that "[d]etermination of obviousness can not be based on the hindsight combination of components selectively culled from the prior art to fit the parameters of the [] invention." *ATD Corp. v. Lydall, Inc.*, 159 F.3d 534, 536 (Fed. Cir. 1998).

Upon application of the law, it appears the Examiner position of obviousness is nothing more than the cautioned-against selective culling from the references in an attempt to fit the limitations of the claims. Not that the Examiner has fit the limitations, but rejections of this sort are clearly discouraged.[5]

---

[5] As is well established, "virtually all [inventions] are combinations of old elements." *Ruiz v. A.B. Chance Co.*, 69 USPQ2d 1686, 1690 (Fed. Cir. 2004). Also, an obvious determination under 35 U.S.C. 103(a) requires an "as a whole" analysis of the prior art to otherwise prevent an impermissible "evaluation of the invention part by part." *Id.* For otherwise, "an obviousness assessment might break an invention into its component parts (A+B+C), then find a prior art reference containing A, another containing B, and another containing C, and on that basis alone declare the invention obvious." *Id.* In turn, "this form of hindsight reasoning, using the invention as a roadmap to find its prior components, would discount the value of combining various existing features or principles in a new way to achieve a new result - often the very definition of invention." *Id.*

D. The Examiner Fails to Meet his Burden of Establishing Obviousness

As longstanding precedent, the initial burden of establishing a prima facie basis to deny patentability to a claimed invention on any ground is always on the examiner. *In re Oetiker*, 977 F.2d 1443, 1445, 24 USPQ2d 1443, 1444 (Fed. Cir. 1992). However, it appears the Examiner's legal position in the instant matter relates exclusively to Slivka teaching/disclosing the entirety of all the pending claim elements with the exception that Angelo provides the teaching of "one of randomly and substantially each time the executable code is launched for use, calculating a plurality of subsequent scores on the executable code." *4-6-2006 Final Office Action, p. 3, second paragraph.* For several reasons, this rationale is flawed and insufficient.

First, the Examiner asserts the motivation or suggestion to combine Slivka and Angelo relates to the nature of the problem to be solved. Namely, the motivation to combine relates to "providing a safestart everytime executing a program [sic]." *4-6-2006 Final Office Action, p. 3, final line.*

Slivka, however, already accounts for its used-based scenarios of computer mode operation and need not look to Angelo for any reason, much less its teaching of a safestart.

Second, the Appellant agrees the law allows for examining the nature of the problem to be solved when determining motivation.[6] However, the instant invention does not simply address solving a problem of making sure safestart modes of operation have no integrity issues. Rather, the instant invention broadly relates software products and executable code fully anticipated of being used during other than safestarts. To this end, it offers a solution to a problem contemplative of regular or routine modes of operation, because hackers will

---

[6] "A suggestion or motivation to modify prior art teachings may appear in the context of the public prior art, in the nature of the problem addressed by the invention, or even in the knowledge of one of ordinary skill in the art." *Underlining added, Princeton Biochemicals, Inc. Beckman coulter, Inc.*, 04-1493, 6/9/2005, 411 F.3d 1332 (Fed. Cir. 1995).

not limit themselves to attacking software products only when computers are being safely started.

It is, therefore, overstated to simply characterize the nature of the problem to be solved as relating to "providing a safestart." Claims 8-16 and 22 also negatively disclaim safestarts by reciting "independence" of system management modes of operation.

Third, the Court of Appeals for the Federal Circuit has warned that, "***simply identifying all of the elements in a claim in the prior art does not render a claim obvious.***" *Ruiz*, 357 F.3d at 1275. Some quantum of proof is certainly required. *Id.* To the extent the Examiner has identified all the elements of any one claim, although the Appellant strongly asserts this has not occurred, the Examiner has so far only offered proof of motivation to combine Slivka and Angelo by contending (without substantiation) that "it would have been obvious at the time the invention was made for one having ordinary skill in the art to modify Slivka's invention to incorporate Angelo secure executing the executable code everytime [sic] it is being launched or requested with a motivation of providing safestart everytime [sic] executing a program." *4-6-2006 Final Office Action, p. 3, final paragraph.* However, this assertion is inaccurate, from above, and rawly given. It merely represents what the Examiner thinks a skilled artisan would have thought about the instant invention over five years ago to the extent Angelo, for example, is even at all relevant to the instant invention or to Slivka. It is also a scant assertion with little, if any, underlying support.

E. Conclusion

The Appellant submits that (1) all claims are in a condition for allowance; (2) that the combination of Slivka and Angelo does not render the pending claims obvious; and (3) that Slivka and Angelo are improperly combined thereby failing the obligation of a *prima facie* position. It is respectfully requested that the rejections of the pending claims be reversed and

the application remanded to the Examiner for allowance. To the extent any fees are due beyond those authorized in the fee transmittals for filing a Notice of Appeal and brief in support thereof, under 37 C.F.R. §§41.20(b)(1) and (b)(2), respectively, the undersigned authorizes their deduction from Deposit Account No. 11-0978.

Respectfully submitted,

KING & SCHICKLI, PLLC

Michael T. Sanderson
Reg. No. 43,082

247 North Broadway
Lexington, KY 40507
(859) 252-0889

Certificate of Mailing

I hereby certify that this correspondence having ______ total pages is being deposited with the United States Postal Service as first class postage pre-paid mail in an envelope addressed to: Mail Stop Appeal Brief - Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 14, 2006

Date 8/14/06 by ________________.

VIII. CLAIMS APPENDIX

The claims on Appeal include 1-4 and 7-22. Of those, claims 1-3, 7, 8, 11-14 and 16-22 appear as previously presented while all others remain as originally presented. Claims 5 and 6 remain canceled.

Listing of Claims:

1. (Previously Presented) A method for validating executable code resident in an operating system having executable instructions, comprising the steps of:

identifying an executable code having an unaltered size;

calculating an initial score associated with the executable code when the executable code is initially or shortly thereafter loaded into an operating system;

saving the initial score;

one of randomly and substantially each time the executable code is launched for use, calculating a plurality of subsequent scores on the executable code;

exclusively comparing each of the subsequent scores to the saved initial score and to no other scores;

if the each of the subsequent scores do not vary from the saved initial score, concluding the executable code maintains the unaltered size; and

if any of the subsequent scores vary from the saved score, concluding the executable code has an altered size.

2. (Previously Presented) The method of claim 1, further comprising the steps of:

unloading the executable code from the operating system if the saved initial score is not equal to the any of the subsequent scores.

3. (Previously Presented) The method of claim 1, further comprising the steps of:

disabling at least a portion of the executable code if the saved initial score is not equal to the any of the subsequent scores.

4. (Original) The method of claim 1, wherein the scores are the result of a checksum calculation.

5. (Canceled)

6. (Canceled)

7. (Previously Presented) The method of claim 1, further comprising the steps of:

notifying electronically an owner of the executable code if the saved initial score is not equal to the any of the subsequent scores.

8. (Previously Presented) A method for disabling executable code which has been modified without authorization having executable instructions, comprising the steps of:

identifying an executable code in an operating system having an unaltered format;

calculating a score associated with the executable code exclusively within an operating system of a computing device independent of a system management mode of operation;

one of randomly and substantially each time the executable code is launched for use, calculating subsequent scores associated with the executable code;

determining the executable code has an altered format if the score is not equal to any of the subsequent scores, the determining exclusively including comparing each of the subsequent scores to the score with no other score comparisons occurring; and

disabling the executable code if the score is not equal to any of the subsequent scores.

9. (Original) The method of claim 8, further comprising the steps of:

notifying an owner of the executable code if disabled.

10. (Original) The method of claim 8, wherein the scores are the result of a checksum calculation.

11. (Previously Presented) The method of claim 8, wherein the subsequent scores are calculated randomly and substantially each time the executable code is launched for use.

12. (Previously Presented) The method of claim 11, wherein the subsequent scores are further calculated at one or more predetermined time intervals.

13. (Previously Presented) The method of claim 8, further comprising the steps of:

removing the executable code if disabled from a memory of the operating system wherein the executable code resides.

14. (Previously Presented) The method of claim 8, further comprising the steps of:

assisting in the loading of the executable code, if not disabled, to a memory of the operating system wherein the executable code resides.

15. (Original) The method of claim 8, further comprising the steps of:

registering the executable code if not disabled; and recording a history if the executable code is disabled.

16. (Previously Presented) A method of authenticating executable code resident in a memory having executable instructions, comprising the steps of:

identifying an executable code having an unaltered format;

acquiring a score associated with an executable code which was established when the executable code was first loaded into a memory of an operating system;

one of randomly and substantially each time the executable code is launched for use after the executable code is said first loaded, receiving subsequent scores on the executable code while the executable code is in the memory;

exclusively comparing the subsequent scores to the score with neither the subsequent scores nor the score being compared to any other values;

if the subsequent scores do not vary from the score, executing the executable code in the unaltered format; and

if any of the subsequent scores vary from the saved score, indicating the executable code has an altered format.

17. (Previously Presented) The method of claim 16, further comprising the steps of:

disabling the executable code while the executable code is in the memory when the any of the subsequent scores is not equal to the score.

18. (Previously Presented) The method of claim 16, further comprising the steps of:

suspending one or more operations of the executable code while the executable code is executing in the memory when the any of the subsequent scores is not equal to the score.

19. (Previously Presented) The method of claim 16, wherein the subsequent scores are received each time the executable code is initiated in the memory for an execution.

20. (Previously Presented) The method of claim 16, reporting one or more system events and variables when the any of the subsequent scores is not equal to the score.

21. (Previously Presented) Functional data used to validate executable code embodied in a computer readable medium, the data comprising:

an unaltered original format;

a first score associated with an executable code when the executable code is initially loaded into an operating system;

a plurality of second scores associated with the executable code at one of 1) a random period of time subsequent to when the executable code was initially loaded and 2) substantially each time the executable code is launched for use subsequent to when the executable code was initially loaded and each second score being operable to be exclusively

compared with the first score to determine if the executable code has been altered since the initial load; and

an altered unoriginal format if the first and second scores do not equal.

22. (Previously Presented) A system for validating executable code, comprising:

an executable code having an unaltered size;

a scoring set of executable instructions operable to receive and record a score associated with the executable code when the code is initially loaded into a computer readable medium;

a comparing set of executable instructions in an operating system of a computing device independent of a system management mode of operation operable to receive subsequent scores associated with the code and to one of randomly and substantially each time the executable code is launched for use exclusively compare the score and the subsequent scores to determine if the code has been altered; and

an executable code having an altered size if the score and the subsequent scores do not equal.

IX. EVIDENCE APPENDIX

None

X. RELATED PROCEEDINGS APPENDIX

None

PTO/SB/17 (10-04v2)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2005

Effective 10/01/2004. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 500.00

Complete if Known	
Application Number	09/862,828
Filing Date	05/22/2001
First Named Inventor	Neil W. Taylor
Examiner Name	Linh L D Son
Art Unit	2135
Attorney Docket No.	1363-004

METHOD OF PAYMENT *(check all that apply)*

☐ Check ☐ Credit card ☐ Money Order ☐ Other ☐ None

☑ Deposit Account:

Deposit Account Number: 11-0978

Deposit Account Name: KING & SCHICKLI, PLLC

The Director is authorized to: *(check all that apply)*

☑ Charge fee(s) indicated below ☑ Credit any overpayments

☑ Charge any additional fee(s) or any underpayment of fee(s)

☐ Charge fee(s) indicated below, **except for the filing fee**

to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Fee ($)	Small Entity Fee Code	Fee ($)	Fee Description	Fee Paid
1001	790	2001	395	Utility filing fee	
1002	350	2002	175	Design filing fee	
1003	550	2003	275	Plant filing fee	
1004	790	2004	395	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	

SUBTOTAL (1) ($)

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

			Extra Claims		Fee from below		Fee Paid
Total Claims		-20** =		X		=	
Independent Claims		- 3** =		X		=	
Multiple Dependent						=	

Large Entity Fee Code	Fee ($)	Small Entity Fee Code	Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	88	2201	44	Independent claims in excess of 3
1203	300	2203	150	Multiple dependent claim, if not paid
1204	88	2204	44	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($)

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Fee ($)	Small Entity Fee Code	Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	430	2252	215	Extension for reply within second month	
1253	980	2253	490	Extension for reply within third month	
1254	1,530	2254	765	Extension for reply within fourth month	
1255	2,080	2255	1,040	Extension for reply within fifth month	
1401	340	2401	170	Notice of Appeal	
1402	340	2402	170	Filing a brief in support of an appeal	500.00
1403	300	2403	150	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,370	2453	685	Petition to revive - unintentional	
1501	1,370	2501	685	Utility issue fee (or reissue)	
1502	490	2502	245	Design issue fee	
1503	660	2503	330	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	790	2809	395	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	790	2810	395	For each additional invention to be examined (37 CFR 1.129(b))	
1801	790	2801	395	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) ____________

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) ($) 500.00

SUBMITTED BY				(Complete *(if applicable)*)	
Name *(Print/Type)*	MICHAEL T. SANDERSON	Registration No. *(Attorney/Agent)*	43,082	Telephone	859.252.0889
Signature	[signature]			Date	8-14-06

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

This collection of information is required by 37 CFR 1.17 and 1.27. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.